

Virtual Dispatch Alternatives to Vtables

25 September, 2001

George Bosworth
CLR Software Architect

Agenda

- **Observations and Biases**
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurements
- Observations

Observations and Biases

- Delayed binding is in general a good thing
 - ◆ loose coupling, resilience, reuse, ...
- Vtables are a bad thing in the large
 - ◆ space inefficient, tight bindings, ...
- Heuristics can mask a lot of ignorance and provide for the unexpected

Agenda

- Observations and Biases
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurements
- Observations

Current Approach (V1)

- Method table contain
 - ◆ vtables for virtual functions
 - ◆ vtables for interface functions
- Interface Table maps interfaces for a type to the interface vtable in a method table

Method table

interface I1 {m1(); }

interface I2 {m2(); m3(); }

class A : I1 {m1(){..}; m2(){..}; }

A.m1, I1.m1

A.m2

class B : A, I2 {m3(){..}; }

A.m1, I1.m1

A.m2, I2.m2

B.m3, I2.m3

class C : B { I2.m2() {..}; }

A.m1, I1.m1

A.m2

B.m2

I2.m2

I2.m3

Method table

interface I1 {m1(); }

interface I2 {m2(); m3(); }

class A : I1 {m1(){..}; m2(){..}; }

A.m1, I1.m1

A.m2

class B : A, I2 {m3(){..}; }

A.m1, I1.m1

A.m2, I2.m2

B.m3, I2.m3

class C : B { I2.m2() {..}; }

A.m1, I1.m1

A.m2

B.m3

I2.m2

I2.m3

Calling Sequences

■ virtual call: c.m2()

```
;virtual invocation of m2(), ecx = this ptr  
mov  eax,[ecx]    ;get method table  
call [eax+4]      ;dispatch thru virtual slot 1
```

■ interface call: i2.m2()

```
;interface invocation of i2.m2(), ecx = this ptr  
mov  eax,[ecx]    ;get method table  
mov  eax,[eax+xx] ;get the interface map for class C  
mov  eax,[eax+yy] ;get the dispatch vtable of i2 in C  
                        ; which is virtual slot 3 in C  
call [eax+0]      ;dispatch thru slot 0 of i2 in C
```


Interface Map

- Tightly packed two dimensional array
 - ◆ each class points to its row
 - ◆ interface ids are monotonically increasing global index

Method Table Statistics

program	Size in bytes						
	# MT	MT	VT	Static	Field	GC	I/Fmap
helloworld.exe	269	53076	21788	14352	636	1688	2776
vswinformsdesigner.	2072	464008	236440	60860	8716	21144	25680
vsstartuppropertygri	761	185328	91512	39688	3684	6072	10888
winformsstarupcomp	705	159728	72340	39008	3604	5748	8008
xsitest.exe	647	115400	48612	25260	2688	5228	5144
oledbtest.exe	987	182044	77616	40840	3764	7860	8536

Winforms Hierarchical Slice

Size in bytes

class	MT	✓	Static	Field	GC	map	parent
Object	100	16	40	0	0	0	
MarshalByRefObject	124	28	40	0	12	3	System.Object[mscorlib.dll]
Component	156	52	20	4	20	16	System.MarshalByRefObject
Control	3664	1288	1772	416	28	136	System.ComponentModel
ScrollableControl	1616	1280	116	4	36	136	System.Windows.Forms.Control
ContainerControl	1640	1316	88	4	44	144	System.Windows.Forms.ScrollableControl
Form	2584	1380	716	248	52	144	System.Windows.Forms.ContainerControl
SplitForm	1940	1388	240	64	60	144	System.Windows.Forms.Form

Type and Slot Usage

Followwork 1						
threshold	VM	Used	Slots	Used VM slots	Crash	
1	104	8	396	8	68	41
	104	4	96	11	46	71
4	104	1	096		9	
8	104	1	096	1	9	8

threshold: number of method tables loaded

VM: number of method tables with at least one slot used (the table that

exceeded the current threshold limit)

Used: number of slots used in all loaded method tables

Used VM slots: number of slots that were freed through garbage collection

Crash: this is not the same as the number of times a method implementation was

crashed. It is the number of times a method implementation was

crashed for reasons that caused a crash

Crash: this is not the same as the number of times a method implementation was

crashed. It is the number of times a method implementation was

crashed for reasons that caused a crash

Crash: this is not the same as the number of times a method implementation was

crashed. It is the number of times a method implementation was

crashed for reasons that caused a crash

Crash: this is not the same as the number of times a method implementation was

crashed. It is the number of times a method implementation was

crashed for reasons that caused a crash

Crash: this is not the same as the number of times a method implementation was

crashed. It is the number of times a method implementation was

crashed for reasons that caused a crash

A bigger program

viewinformdesigner

threshold	RA	Used	#slots	Used#	v-slots	RealR
1	100%	440	57510	17,1	1,003	174106
	100%	412	57510	2874	21664	2173538
2	100%	369	57510	14	0406	111893
8	100%	340	57510	100	14000	180000
16	100%	300	57510	1341	1000	10500
32	100%	290	57510	940	17100	215007
64	100%	17	57510	032	15805	2148738
128	100%	180	57510	013	1365	111014
256	100%	144	57510	455	0514	111110
512	100%	108	57510	332	033	2050404
1024	100%	1	57510	004	410	000000
2048	100%	44	57510	107	133	030000
4096	100%	38	57510	6	1103	11111
8192	100%	22	57510	43	10	510000
16384	100%	11	57510	00	400	1000000
32768	100%	11	57510	10	430	130000

Observations

- Most invocations go thru few slots
- Not many types actually instantiated
- Something weird going on in Winforms
 - 11 slots account for 5% the calls
 - 4 of those slots are in HashTable

Other programs



Limitations

- vtables require the code generator (jit) to bind to a slot number
 - cannot decide later (dynamic invocation for scripting)
 - use another binding approach (multimethods)
 - detect out of range (version resiliency, extensibility)
- Must be built whether needed or not

What is needed

- Mapping function from:
 (calling frame, ref token)

- to:
 method implementation address

- Vtables are a restricted form of this mapping function where:

- calling frame is for the `this` type

- ref token is `object` or `member`

- `object` or `member` is

- `object` or `member` is `object` or `member`

Asymmetries to leverage

- Few actual types instantiated
- Small number of slots used
- Assumption we will make:
 - most call sites are mostly monomorphic

Alternative Stub Dispatch

- Type and method specific stubs
- Different kinds of stubs
 - mostly monomorphic
 - polymorphic
- Stubs built on demand as needed
- Heuristics used to choose and adjust the kind of stub in use

Simple Logical Stub Model

- generated call sites:

- aligning pad/nop if necessary (0-3 bytes)
 - call stubaddress

- initial stubaddress points to:

- push \$methodRefToken; identifies contract member
being resolved

- after use, stubaddress points to:

- jump (or jump+offset) starts the return type
jrn fail

- returnMethodAddress

- returnContractRef

if (returnMethodAddress == null) {
 returnContractRef = returnContractRef;
}

Trace based Measurements

- Traces from a modified fjit.
- Stub model consumed traces

`call_cnt` is the running counter of the number of object and interface calls made since the start of the program.

`obj_cnt` is the number of objects created since the program start.

`obj_size` is the number of new objects made since the start of the program.

`obj_size` is the number of objects created since the start of the program. `call_cnt` is the number of times the actual runtime type of the object is not the type of the object being created. `obj_size` is the number of objects created since the start of the program.

`obj_size` is the number of objects created since the start of the program.

(Cont) Trace for OLEDBTEST

Year	1990	1991	1992	1993	1994
1990	0	0	0	0	0
1991	0	0	0	0	0
1992	0	0	0	0	0
1993	0	0	0	0	0
1994	0	0	0	0	0
1995	0	0	0	0	0
1996	0	0	0	0	0
1997	0	0	0	0	0
1998	0	0	0	0	0
1999	0	0	0	0	0
2000	0	0	0	0	0
2001	0	0	0	0	0
2002	0	0	0	0	0
2003	0	0	0	0	0
2004	0	0	0	0	0
2005	0	0	0	0	0
2006	0	0	0	0	0
2007	0	0	0	0	0
2008	0	0	0	0	0
2009	0	0	0	0	0
2010	0	0	0	0	0
2011	0	0	0	0	0
2012	0	0	0	0	0
2013	0	0	0	0	0
2014	0	0	0	0	0
2015	0	0	0	0	0
2016	0	0	0	0	0
2017	0	0	0	0	0
2018	0	0	0	0	0
2019	0	0	0	0	0
2020	0	0	0	0	0
2021	0	0	0	0	0
2022	0	0	0	0	0
2023	0	0	0	0	0
2024	0	0	0	0	0
2025	0	0	0	0	0
2026	0	0	0	0	0
2027	0	0	0	0	0
2028	0	0	0	0	0
2029	0	0	0	0	0
2030	0	0	0	0	0
2031	0	0	0	0	0
2032	0	0	0	0	0
2033	0	0	0	0	0
2034	0	0	0	0	0
2035	0	0	0	0	0
2036	0	0	0	0	0
2037	0	0	0	0	0
2038	0	0	0	0	0
2039	0	0	0	0	0
2040	0	0	0	0	0
2041	0	0	0	0	0
2042	0	0	0	0	0
2043	0	0	0	0	0
2044	0	0	0	0	0
2045	0	0	0	0	0
2046	0	0	0	0	0
2047	0	0	0	0	0
2048	0	0	0	0	0
2049	0	0	0	0	0
2050	0	0	0	0	0
2051	0	0	0	0	0
2052	0	0	0	0	0
2053	0	0	0	0	0
2054	0	0	0	0	0

Summary for OLEDBTEST

fitted sites	3273	called sites	2130
stub calls	864968	misses	24457 (2.8%)
stubs made	897	tokens	121
stubs/site	8.17	stub refs	5.54

- small number of stubs and tokens
- low miss rate
- program has:
 - 987 types
 - over 10,000 methods
 - 20,000 variable slots

Mono or Polymorphic?

missile	calls	sites	misses	status
0-0	750513	30031	100	045
0-10	07040	0	020	050
0-20	2714	0	00	10
0-30	0077	0	000	00
0-40	1993	0	00	00
0-50	0160	0	00	00
0-60	02700	0	000	000
0-70	7040	0	000	000
0-80	0000	0	000	000
0-90	0000	0	000	000
0-100	0000	0	000	000
0-110	0000	0	000	000
0-120	0000	0	000	000
0-130	0000	0	000	000
0-140	0000	0	000	000
0-150	0000	0	000	000
0-160	0000	0	000	000
0-170	0000	0	000	000
0-180	0000	0	000	000
0-190	0000	0	000	000
0-200	0000	0	000	000
0-210	0000	0	000	000
0-220	0000	0	000	000
0-230	0000	0	000	000
0-240	0000	0	000	000
0-250	0000	0	000	000
0-260	0000	0	000	000
0-270	0000	0	000	000
0-280	0000	0	000	000
0-290	0000	0	000	000
0-300	0000	0	000	000
0-310	0000	0	000	000
0-320	0000	0	000	000
0-330	0000	0	000	000
0-340	0000	0	000	000
0-350	0000	0	000	000
0-360	0000	0	000	000
0-370	0000	0	000	000
0-380	0000	0	000	000
0-390	0000	0	000	000
0-400	0000	0	000	000
0-410	0000	0	000	000
0-420	0000	0	000	000
0-430	0000	0	000	000
0-440	0000	0	000	000
0-450	0000	0	000	000
0-460	0000	0	000	000
0-470	0000	0	000	000
0-480	0000	0	000	000
0-490	0000	0	000	000
0-500	0000	0	000	000
0-510	0000	0	000	000
0-520	0000	0	000	000
0-530	0000	0	000	000
0-540	0000	0	000	000
0-550	0000	0	000	000
0-560	0000	0	000	000
0-570	0000	0	000	000
0-580	0000	0	000	000
0-590	0000	0	000	000
0-600	0000	0	000	000
0-610	0000	0	000	000
0-620	0000	0	000	000
0-630	0000	0	000	000
0-640	0000	0	000	000
0-650	0000	0	000	000
0-660	0000	0	000	000
0-670	0000	0	000	000
0-680	0000	0	000	000
0-690	0000	0	000	000
0-700	0000	0	000	000
0-710	0000	0	000	000
0-720	0000	0	000	000
0-730	0000	0	000	000
0-740	0000	0	000	000
0-750	0000	0	000	000
0-760	0000	0	000	000
0-770	0000	0	000	000
0-780	0000	0	000	000
0-790	0000	0	000	000
0-800	0000	0	000	000
0-810	0000	0	000	000
0-820	0000	0	000	000
0-830	0000	0	000	000
0-840	0000	0	000	000
0-850	0000	0	000	000
0-860	0000	0	000	000
0-870	0000	0	000	000
0-880	0000	0	000	000
0-890	0000	0	000	000
0-900	0000	0	000	000
0-910	0000	0	000	000
0-920	0000	0	000	000
0-930	0000	0	000	000
0-940	0000	0	000	000
0-950	0000	0	000	000
0-960	0000	0	000	000
0-970	0000	0	000	000
0-980	0000	0	000	000
0-990	0000	0	000	000
0-1000	0000	0	000	000

0-1000

0-1000

0-1000

0-1000

0-1000

0-1000

Modified CLR built

- EE generates and manages stubs
- Jit generates stub calls for interface and virtual methods
- What wasn't done:
 - eliminate the vtables themselves
 - integrate stubs with app domain unloading
 - atomic update of call sites partially undone

Tokens

- tokens composed of two 32bit fields
 - ◆ contract
 - 0 means use the type of «this»
 - 0 means contract is the interface id
 - ◆ member
 - slot number in the table or interface table
- Since the numbers are small, it is usually (always) bit encoded in 31 bits

Call Sites

■ with call site rewriting

```
mov eax,[ecx] //preload the MT  
call stub //direct call to stub
```

■ as pure code

```
mov eax,[ecx] //preload the MT  
call [stubcall] //indirect call to stub
```

Lookup Stubs

- Call sites initially point to:

- push token

- jmp ResolveWorkerStub

- ResolveWorkerStub enters the EE and causes a Dispatcher stub to be found (or created) that is specific to the type of the `this` pointer of the call
 - betting monomorphic until shown otherwise

Dispatcher Stubs

```
cmp eax,expectedMT  
jne failstub; must be fwd branch  
jmp targetaddress
```

- Checks that the actual type of «this» is what is expected, and if so transfers to the method implementation
- fail stub will cause a lookup to be done and then pass control to the correct method.
- One per tuple (expectedMT, token)

Fail Stub

```
sub [counter], 1
```

```
if [patchstub] ; must be fwd jmp
```

```
fall into resolve stub
```

- Fail stub checks to see if we are failing a lot, counter is periodically inc'd
- Patch stub will patch the call site to point to the resolve stub
- Resolve stub will do a cache lookup
- One per token, 1 dispatch stub share

Patch Stub

call PatchWorkerStub

jmp resolvestub

- PatchWorkerStub goes into the EE and decides whether or not the call site should be patched to point to the resolvestub directly, i.e. make it a polymorphic call site.

Resolve Stub

[illegible]

Resolve Stub

```

main() {
    char s[100];
    int i;
    while(s[i]!='\n')
        i++;
    printf("%d\n", i);
}

```

Resolve Stub (Cont)

- Resolve stubs to a cache lookup keyed by the type of <this> and the token.
- Cache contains the method impl address.
- Key to the cache hit rate is the hash fn used
 - current has not been extensively tuned

Agenda

- Observations and Biases
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurement
- Observations

Simple calls - best case

nested calls embedded in a loop

Loop is just the loop

	loop count	current jit	1 static	1 static
Loop	1000000	0	0	0
Loop	1000000	0	3	0
Loop	10000000	4%	4%	3%
Call	100000	1%	1%	1%
Call	1000000	1.2%	1.2%	1.2%
Call	10000000	16.8%	16.2%	16.8%
Static	1000000	1%	1%	1%
Static	1000000	1.2%	1.2%	1.8%
Static	10000000	16.2%	16.2%	16.8%
Interface	100000	1%	1%	0
Interface	1000000	1.9%	1.9%	1.9%
Interface	10000000	19.3%	18.7%	14.6%
Virtual	100000	1%	1%	0
Virtual	1000000	1%	1%	5.0%
Virtual	10000000	16%	15.5%	14.0%

The bad news

- The timings are very unstable. Small changes in unrelated portions of the EE can cause big jumps (factor of 2) in the above numbers.
- Tried, unsuccessfully, to isolate via vtune.
- Built a vtune post processor that computed aggregate stats for the stubs themselves, still didn't isolate.

Controlled Sequences

- Built a hardcoded set of code sequences that did the current virtual dispatch and the stub dispatch
- Varied the calling and the called environment
- Basically a loop of 20,000 around 50 inlined call sites

Controlled Sequences (Cont)

- NOP(s) of 1 or 2 bytes are inserted between every 10 calls
- (CONTEXT 0) = no inter call instr overhead
- (CONTEXT 5) = 5 instr (dummy bp frame-like) inter call overhead
- Called routine was just a return, and was either Frameless or EBP Framed

Controlled Sequences (Cont)

		4400	4401	4402	4403
Small	Performance	1.1	1.1	1.1	1.1
MA	4400	4.411		1.1	1.1
MA	4400/4401	1.1	1.1	1.1	1.1
MA	4400/4402	1.1	1.1	1.1	1.1
MA	4400/4403	1.1	1.1	1.1	1.1

- anomalies make one read the intel performance manuals really close

does it vary with code size?

[illegible]

- Small - loop 20,000 over 50 inlined sites
- Large - loop 1,000 over 1,000 inlined sites

Say hello to the BTB

■ Branch Transfer Buffer on the Pentium

- all my runs were on a 933 Mhz PIII with 800Mhz RDRAM

Branch basics on pentiums

- The BTB is checked on every instr fetch of any kind. If it hits, it turns all transfers into zero or near zero cost
- If no BTB hit, fall thru is free
- If non-fall thru but statically predicted, e.g. a jmp relative, cost is a half dozen or so cycles
- If not statically predicted, e.g. an indirected call or a mis-prediction cost is a score or more cycles and this doesn't include any cache or memory fetches at all
- BTB = 32K entries or so. Not small

Simple Calls revisited

姓名	性别	年龄	职业	住址	联系电话	电子邮箱	身份证号	银行卡号	支付宝账号	微信账号	其他联系方式
张三	男	25	程序员	北京市朝阳区	13800138000	zhangsan@163.com	110101199801010001	62284801010101010101	15888888888	zhangsan	15888888888
李四	女	30	设计师	北京市海淀区	13900139000	lisi@163.com	110102196802020002	62284801010101010101	15888888888	lisi	15888888888
王五	男	35	销售经理	上海市浦东新区	13700137000	wangwu@163.com	310101196303030003	62284801010101010101	15888888888	wangwu	15888888888
赵六	女	28	教师	广东省广州市	13600136000	zhaoliu@163.com	440101197004040004	62284801010101010101	15888888888	zhaoliu	15888888888
孙七	男	40	工程师	浙江省杭州市	13500135000	sunqi@163.com	330101195805050005	62284801010101010101	15888888888	sunqi	15888888888
周八	女	32	会计	江苏省南京市	13400134000	zhouba@163.com	320101196606060006	62284801010101010101	15888888888	zhouba	15888888888
吴九	男	27	产品经理	福建省厦门市	13300133000	wujiu@163.com	350101197107070007	62284801010101010101	15888888888	wujiu	15888888888
郑十	女	38	市场专员	山东省济南市	13200132000	zhengshi@163.com	370101195608080008	62284801010101010101	15888888888	zhengshi	15888888888
陈十一	男	29	数据分析师	河南省郑州市	13100131000	chen11@163.com	410101196909090009	62284801010101010101	15888888888	chen11	15888888888
周十二	女	31	运营专员	湖北省武汉市	13000130000	zhou12@163.com	420101196710100010	62284801010101010101	15888888888	zhou12	15888888888
吴十三	男	33	系统管理员	湖南省长沙市	12900129000	wu13@163.com	430101196511110011	62284801010101010101	15888888888	wu13	15888888888
郑十四	女	36	人力资源	四川省成都市	12800128000	zheng14@163.com	510101195912120012	62284801010101010101	15888888888	zheng14	15888888888
陈十五	男	39	项目经理	贵州省贵阳市	12700127000	chen15@163.com	520101195713130013	62284801010101010101	15888888888	chen15	15888888888
周十六	女	41	财务主管	云南省昆明市	12600126000	zhou16@163.com	530101195514140014	62284801010101010101	15888888888	zhou16	15888888888
吴十七	男	42	IT支持	陕西省西安市	12500125000	wu17@163.com	610101195315150015	62284801010101010101	15888888888	wu17	15888888888
郑十八	女	43	行政助理	甘肃省兰州市	12400124000	zheng18@163.com	620101195116160016	62284801010101010101	15888888888	zheng18	15888888888
陈十九	男	44	销售总监	宁夏回族自治区银川市	12300123000	chen19@163.com	640101194917170017	62284801010101010101	15888888888	chen19	15888888888
周二十	女	45	法务专员	新疆维吾尔自治区乌鲁木齐市	12200122000	zhou20@163.com	650101194718180018	62284801010101010101	15888888888	zhou20	15888888888

Indirections, like stable dispatch, fall apart in the large.

Agenda

- Observations and Biases
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurements
- Observations

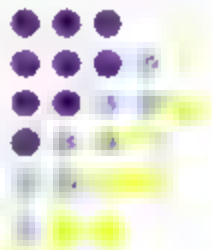
Conclusion

- Stubs can be performant
- Perf is hard to measure
- Things like BTB make tuning almost impossible
- Transfers of control carry a large hidden perf cost on Pentiums, since any transfer will put pressure on the BTB

See <http://www.sds.tamu.edu/engr/engr539/engr539.html>

Continuations, anyone?

Nick Benton
MSR Cambridge





What's a continuation?

- The rest of the computation
- Having first class continuations in a language or VM means being able to *reify* the current rest of the computation into a value which can then be passed around and invoked later
 - Possibly never
 - Possibly several times
- To a first approximation grab the stack and turn it into an object on the heap
- This single facility allows one to build lots of fancy control structures which would normally be implemented separately



For example

- Exceptions
- Backtracking a la Prolog
- Coroutines
- Lightweight threads (CML)
- Stateless servers
- Mobile computations (migratory apps)
- Security features
- Time-travel debugging
- Arbitrary monads – adding step-counting
instrumentation non-determinism probabilistic
computation



Monadic reflection using call/cc

- Add nondeterminism (for example) to already-compiled code

```
let cc = call/cc (lambda (cc)
  (lambda (f)
    (lambda (x)
      (f (cc) x)))))

let f = (lambda (cc)
  (lambda (x)
    (cc) x))

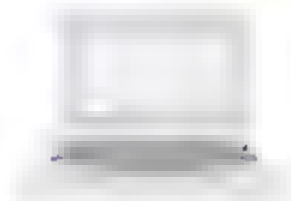
let g = (lambda (cc)
  (lambda (x)
    (cc) x))

...

```

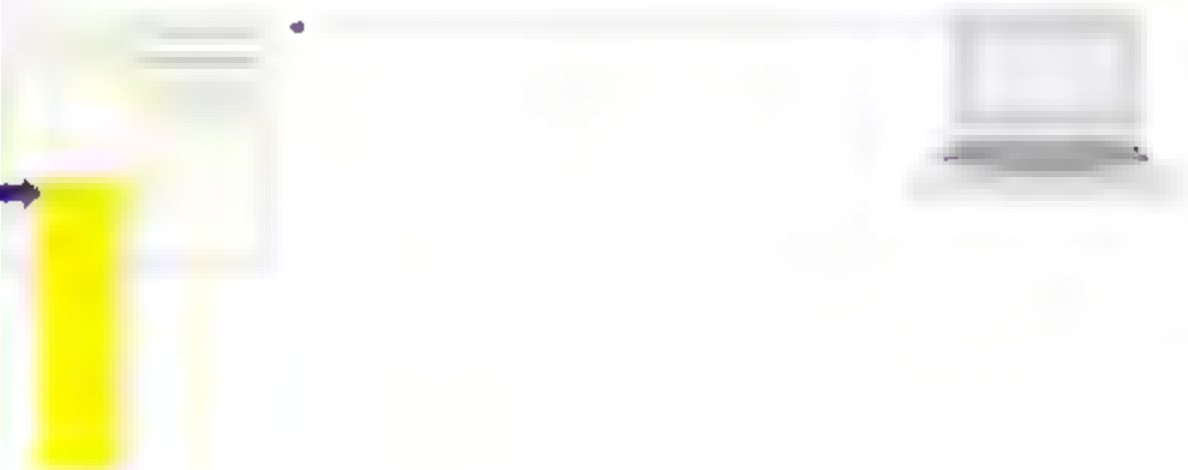



e.g. stateless servers



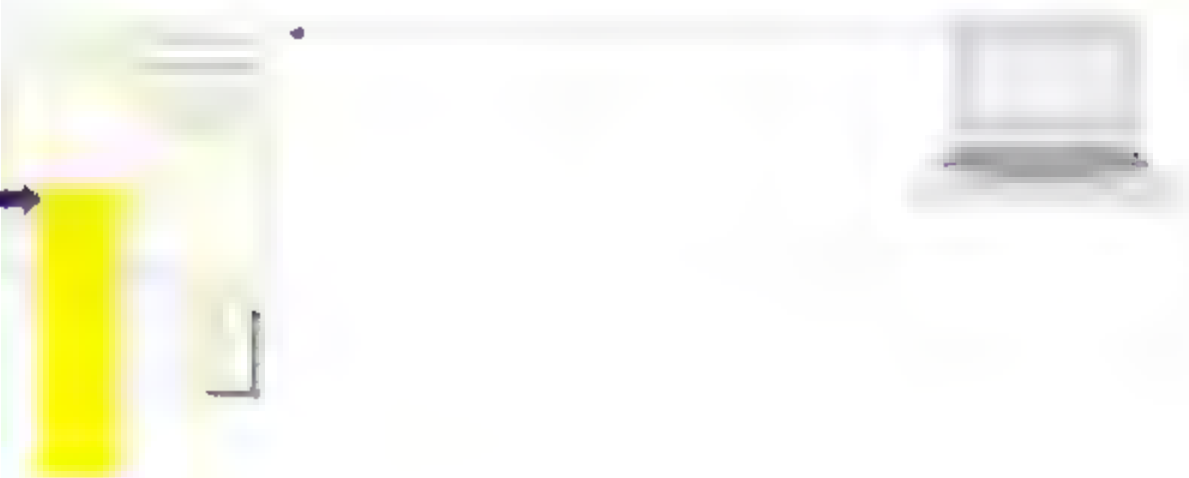


e.g. stateless servers



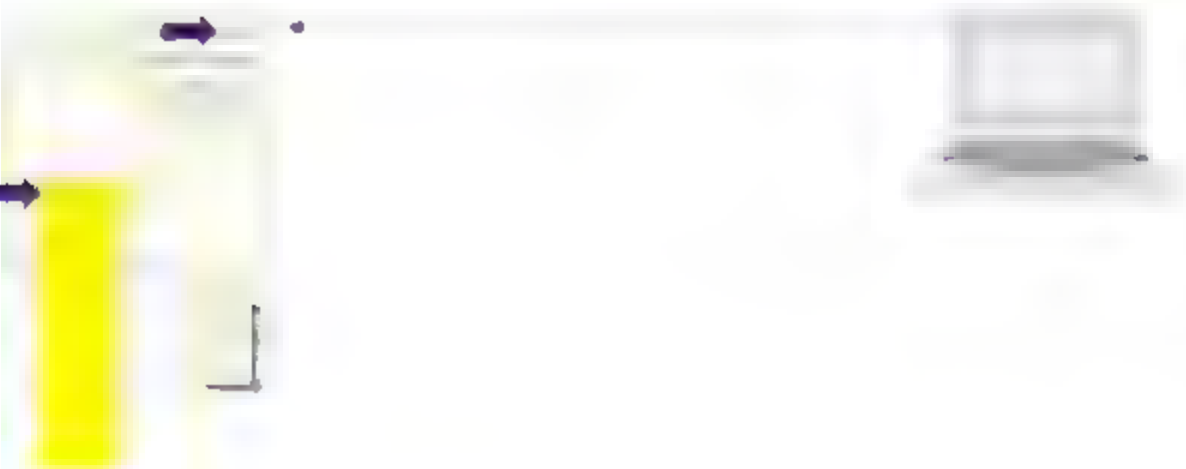


e.g. stateless servers





e.g. stateless servers





History

- First invented in denotational semantics in order to give a mathematical account of the meaning of existing control operators, such as jumps
- Then added to programming languages such as Scheme and SML/NJ (call/cc)
- Still interesting research going on
 - More refined control operators (prompt/reset hierarchies)
 - Connection with classical logic



Continuation Passing Style

- There's a uniform translation of CBV (or, indeed, CBN) into CPS
 - Every function gets an extra argument "what to do with the result"
 - Every call becomes a tail call

Why should a VM support first-class continuations?



- Some languages have them and compiling via explicit CPS to the runtime gives poor performance
- Similarly some other languages have features (e.g. backtracking, large numbers of very lightweight threads) that can be implemented more efficiently with continuations
- It's potentially simpler than adding lots of control-related stuff in an ad hoc way
- A general purpose runtime should provide general purpose mechanisms

Why should a VM support first-class continuations?



- Some languages have them and compiling via explicit CPS to the runtime gives poor performance
- Similarly some other languages have features (e.g. backtracking, large numbers of very lightweight threads) that can be implemented more efficiently with continuations
- It's potentially simpler than adding lots of control-related stuff in an ad hoc way
- A general purpose runtime should provide general purpose mechanisms

Why *shouldn't* a VM support first class continuations?



- Scary
- Almost certainly interact badly with existing features (e.g. security)
- Rule out some optimisations which might mean paying an efficiency cost in the common case of languages which don't use them



Conclusion

- None, really
- But it's something that seems worth thinking about

LZ77 data compression

- Before: .
- After' <-5,8>.
- Encoding'
- Recast as "instructions" to a decompressor'
 - ' '
 - ' '
 - ' '
 - ' '
 - ' '
 - ' '
 - ' '

The echo instruction

- Example:

```
ldc.i4 12  
add  
ldind.i4  
...  
echo 30,3
```

- Like `#include` but on-the-fly
- Like JSR but caller delimits entry/exit without overhead polluting the stream

Summary

- Cuts LCC byte-code by ~40% == 0.60x
- Echo suits interpreters:
 - Clients that can't JIT or even decompress
 - e.g., "feature" code in cell phones
- Related tools:
 - Automatic generation of compressed bytecodes
 - High-tech code and XML compression
(esp. for thin or costly pipes, service packs, ROM)
 - Simple, table-driven code generators and JITs



Dynamic Heuristically Driven Speculative Optimizations

George Bosworth